DESIGN PATTERNS

**MVP**

**Model** : Handles the data part of your application

**Presenter** : Acts as a bridge between Model and View.

**View**: A Fragment/Activity that is responsible for Laying out views with the data from presenter

Model and View will never interact.

View : Fragment/Activity/CustomView represents a View:

Presenter: Listening for user interactions from view, updates Model , gets notified from model with updates that are again updated to Views.

public interface LoginContract {

interface LoginView{

void onInputFieldsEmpty();

void onSuccess();

}

interface LoginPresenter{

void performClick(String name,String password);

}

}

Public class MyActivity extends Activity implements LoginContract.LoginView{

}

Public class LoginPresenterImplemetion extends LoginContract.LoginPresenter{

Public LoginPresenter(LoginView loginView) { this.LoginView = loginView}

Public void performClick(String name,String password){

loginView.isInputFieldEmpty()…..

}

}

Observer Design Pattern:

Is a behavioral design patterns where we are interested state of an object and want to get notified when there is change in its state. The object that observes the state of an object are called **Observers** and the object being watched is called **Subject**.

Adapter Design Pattern

Is a structural design pattern where 2 unrelated interfaces can work together.

Class Volt{

Private int vot;

Public volt(int volt){

This.volt = volt

}

Public void setVolt(int volt){ this.volt = volt}

Public int getVolt(){ return volt;}

}

Class Socket{

Public Volt getVolt(){

new Volt(120);

}

}

Class SocketADapter{

Public Volt get3Volts();

Public Volt get10Volts();

Public Volt get120Volts();

}

Public Class SocketImplementor implements SocketAdapter{

Private Socket socket = new Socket();

@override

Public void get120Volts(){

Socket.getVolts();

}

@override

Public void get10Volts(){

return converVolt(Socket.getVolts,10);

}

@override

Public void get3Volts(){

return converVolt(Socket.getVolts,40);

}

private Volt convertVolt(Volt v, int i) {

return new Volt(v.getVolts()/i);

}

}

Public class AdapterTest{

Psvm(){

SocketADapter sockAdapter = new SocketImplementor();

Volt v3 = getVolt(sockAdapter,3);

Volt v12 = getVolt(sockAdapter,12);

Volt v120 = getVolt(sockAdapter,120);

private static Volt getVolt(SocketAdapter sockAdapter, int i) {

switch (i){

case 3: return sockAdapter.get3Volt();

case 12: return sockAdapter.get12Volt();

case 120: return sockAdapter.get120Volt();

default: return sockAdapter.get120Volt();

}

}

}

}